Configuration Manual

MSc Internship
MSc Cyber Security

Yashodha Patil
Student ID: x19102437

School of Computing
National College of Ireland

Supervisor: Prof. Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Miss. Yashodha Subhash Patil……………………………………
Student ID:	X19102437…………………………………………………………………….……
Programme:	MSc Cyber Security…………………… **Year:** 2019-2020…….
Module:	MSc Internship…………………………………………………………….………
Lecturer:	Prof. Vikas Sahni……………………………………..………
Submission Due Date:	17/08/2020……………………………………………………………………….………
Project Title:	Detection of Clickjacking attacks using the Extreme Learning Machine algorithm.
Word Count:	961…………………………………… **Page Count:** 10

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

I agree to an electronic copy of my thesis being made publicly available on NORMA the National College of Ireland's Institutional Repository for consultation.

Signature: ……

Date: ……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Yashodha Patil
Student ID: x19102437

1 Introduction

The configuration manual document contains the information on the tools and technologies used to carry out the implementation of the research project. Section 2 contains introduction to software tools and technologies required for the implementation. Section 3 contains step wise procedure from starting the application till getting the output. The source code for the implementation is also explained.

2 Software Tools and Technologies used

To create the web application, a python Django framework version 2.2.7 and python version 3.7.1 is used.

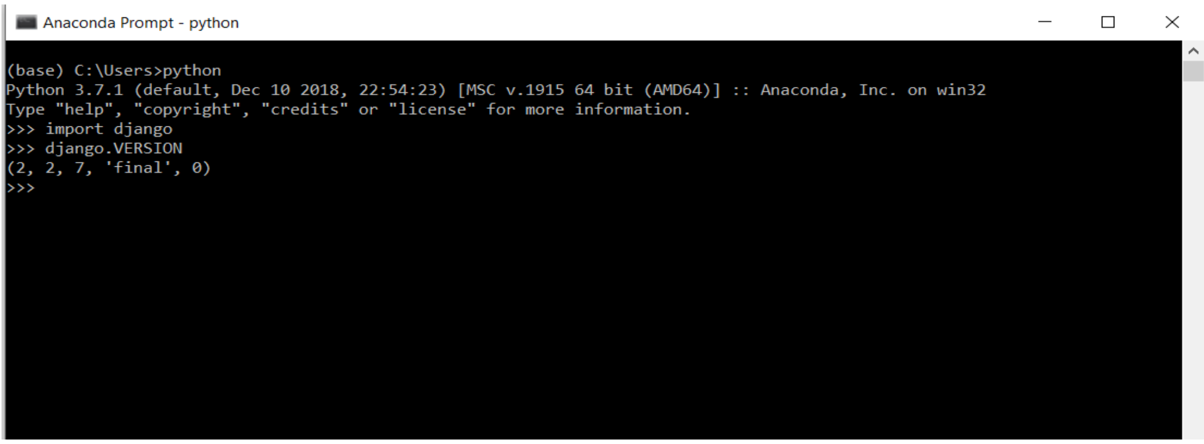

Figure 1: Version of python and Django

The Jupiter notebook which is an open source web application tool is used for data cleaning process and to train the Extreme learning machine algorithm. It supports python programming languages and the HTML components. It gives the interactive output.

Figure 2: Jupyter Notebook

3 Implementation and Deployment steps:

- Step 1: Install the Anaconda navigator from google, to access the Jupiter from this navigator. By clicking on the launch button, the Jupiter notebook will get open in web browser.

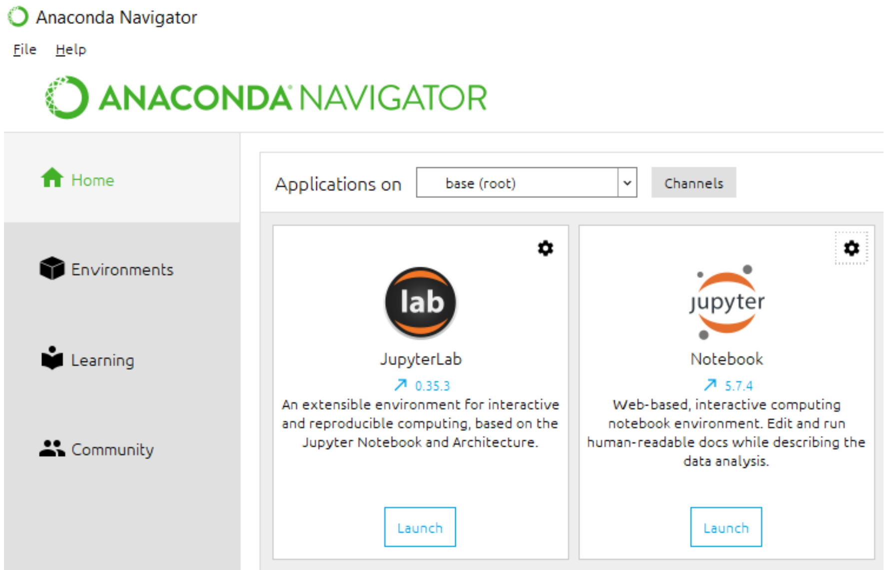

Figure 3 : Install Anaconda Navigator

- Step 2: Open the file from the location where the "ELMClassification.ipynb" file is stored in the Jupiter notebook.

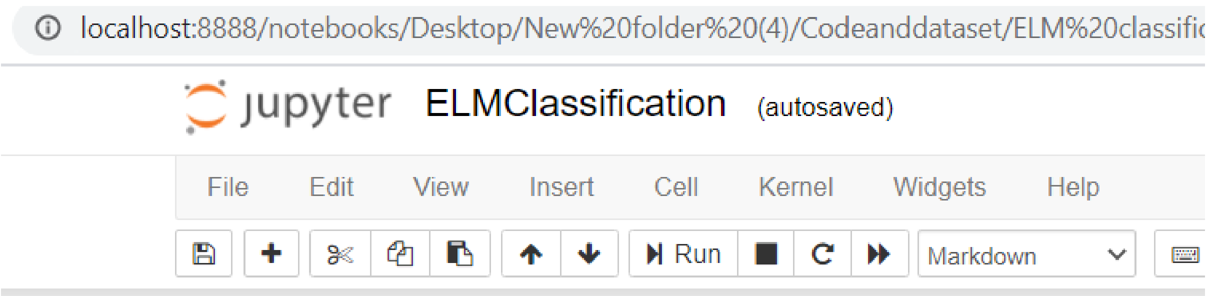

Figure 4 : Launch the jupiter and open the file from the location.

- Step 3: In this file, the steps carried for the Binary classification through ELM model such as,

1) Important libraries are imported such as pd, np, sns, pickle, ELMClassifier, time and many more.

Importing all the neccessary libraries

```
import pandas as pd
import numpy as np
import seaborn as sns
import pickle
import time
from sklearn.svm import SVC
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn_extensions.extreme_learning_machines.elm import ELMClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn import metrics
```

Figure 5: Imported Libraries

2) The dataset is loaded and data processing is performed as shown in the fig:6 and fig:7

loading Dataset

```
data=pd.read_csv("phishing_data.csv").drop(['id'],axis=1)
```

Figure 6: Dataset Loaded

```
def nocaseGraph(data):
    pd.value_counts(data['Result']).plot.bar()
    plt.title('Phishing class histogram')
    plt.xlabel('Class')
    plt.ylabel('Frequency')
    print("-1: Phising cases")
    print("1: Non- Phising cases")
    data['Result'].value_counts()
```

```
def data_analysis(data):
    print(">>>>>>>>>>>>>>>>>>>>>>>>>>> ANALYSIS >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>")

    print("---------------------------------------------------------------------------")
    print("No of rows and columns:",data.shape)
    print("---------------------------------------------------------------------------")
    print("List of column name\n")
    print(data.columns.to_list())
    print("---------------------------------------------------------------------------")
    print("Checking how many columns contain null value",data.isnull().values.any())
    print("---------------------------------------------------------------------------")
    print("Checking whether dataset imbalance or not")
    # Determine number of phishing and non phishing cases in dataset
    nocaseGraph(data)
    non_phishing = data[data['Result'] == 1]
    phishing = data[data['Result'] == -1]
    print('Phishing Cases: {}%'.format(round(len(phishing)/len(data),2)))
    print('Non-Phishing cases: {}%'.format(round(len(non_phishing)/len(data),2)))
    print("##### Data set is mostly balanced, hence no need to balance the dataset #####")
```

Figure 7 : Data Preprocessing

3) Then, the dataset is split into Training and the testing dataset.

spiliting the dataset for training and testing

for testing we take 20%

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=8)
```

Figure 8: Split Dataset

4) The ELM Classifier is trained based on the parameters n_hidden, alpha and the activation_func = ‘tanh’.

```
model=ELMClassifier(n_hidden=150, alpha=0.99, activation_func='tanh')
x_train = np.array(x_train)
y_train = np.array(y_train)
start_elm = time.time()
model.fit(x_train,y_train)
stop_elm = time.time()
print("training time by elm: {} sec".format(round((stop_elm -start_elm),2)))
```

Figure 9 : ELM model Training

5) Now, the Trained ELM model is saved in the pickle file, to use later to classify the links based on this learnings.

```
# now you can save it to a file
with open('elmclassifier..pkl', 'wb') as f:
    pickle.dump(model, f)
```

Figure 10 : converted to pickle file

Step 4: Run the web application through anaconda prompt. Go to the location where the web application folder is stored from the anaconda prompt.

Anaconda Prompt

```
(base) C:\Users\Siddharth Patil>cd C:\Users\Siddharth Patil\Desktop\New folder
(3)\Django webapplication\clickjacking

(base) C:\Users\Siddharth Patil\Desktop\New folder (3)\Django webapplication\cl
ickjacking>
```

Figure 11: Path

Step 5: To run the web application, run the command – python manage.py runsslserver.

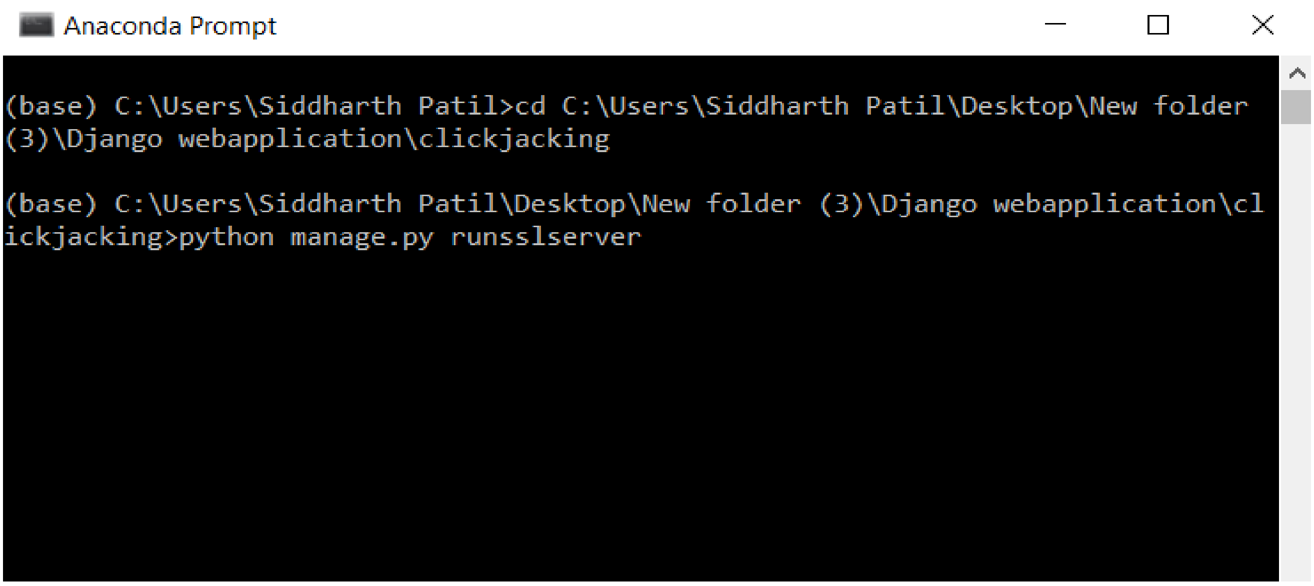

Figure 12: run command

Step 6: From the browser, open the webapplication on localhost – https://127.0.0.1:8000/button

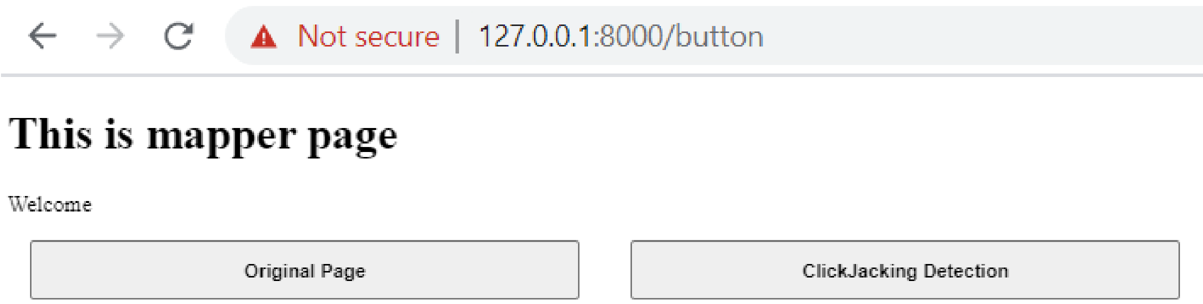

Figure 13: Web application

Step 7: Click on original page button, it will redirect to original page of the webapplication.

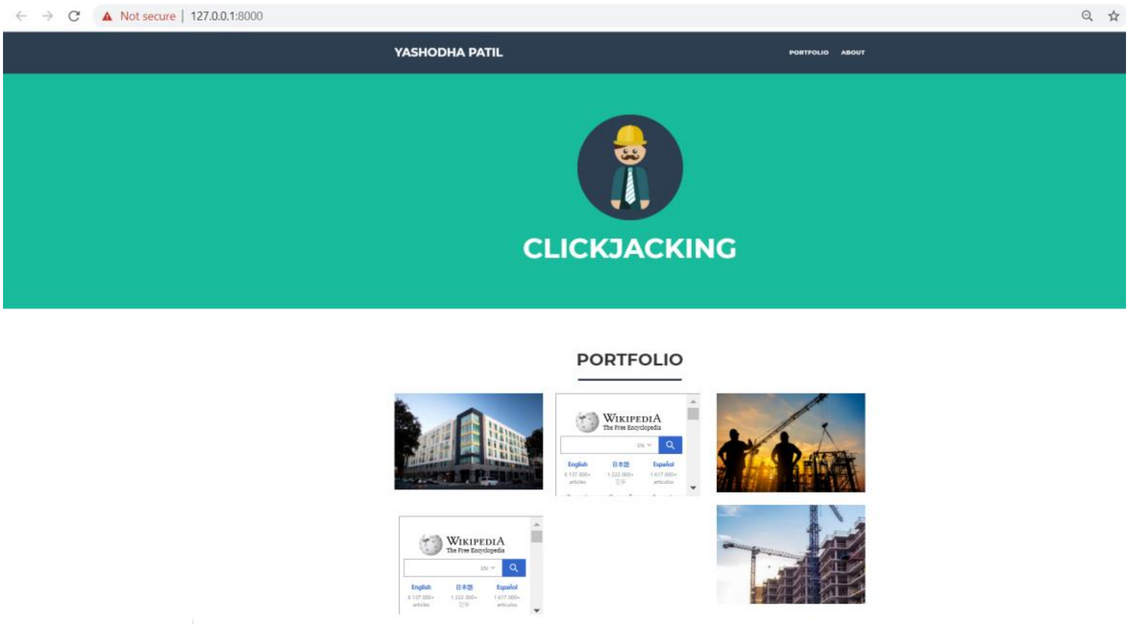

Figure 14: Original Web application page

Step 8: Click on original page button, it redirects to the dummy page of the web application, where all the hidden links and iframe are made visible.

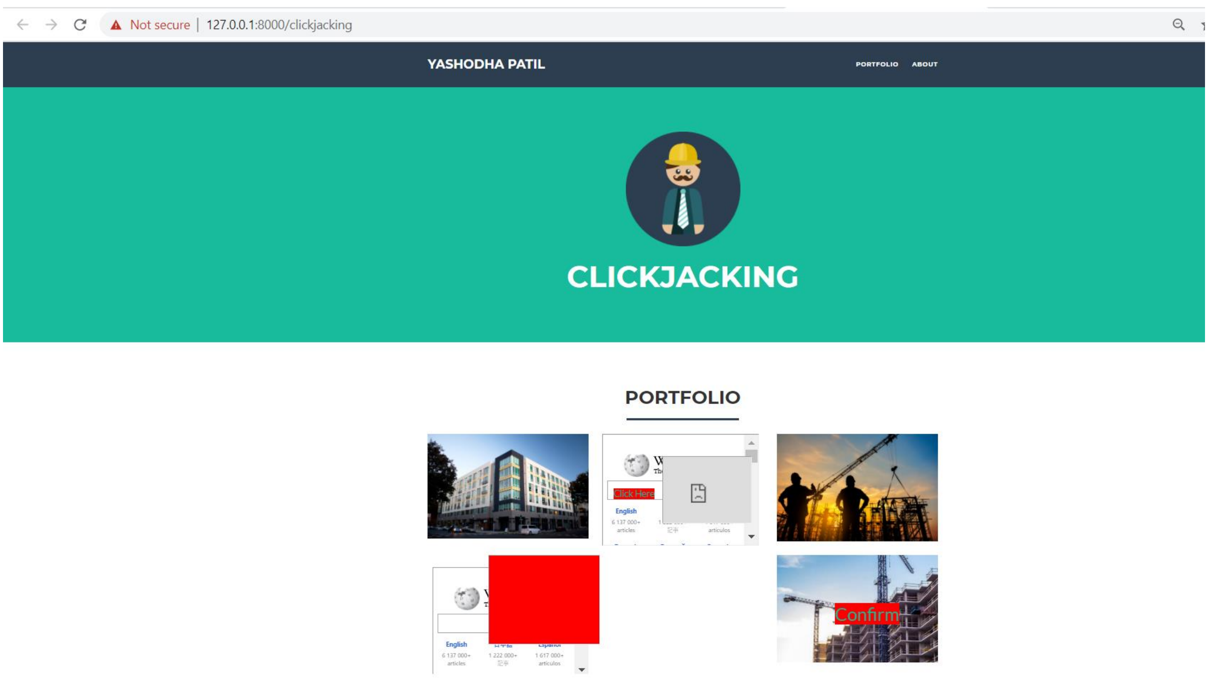

Figure 15: Dummy web page – clickjacked page

Step 9: To view the source code written for the creation of dummy HTML page, open the Django web application project in pycharm and open the Extractor.py file from djangonautic sub menu.

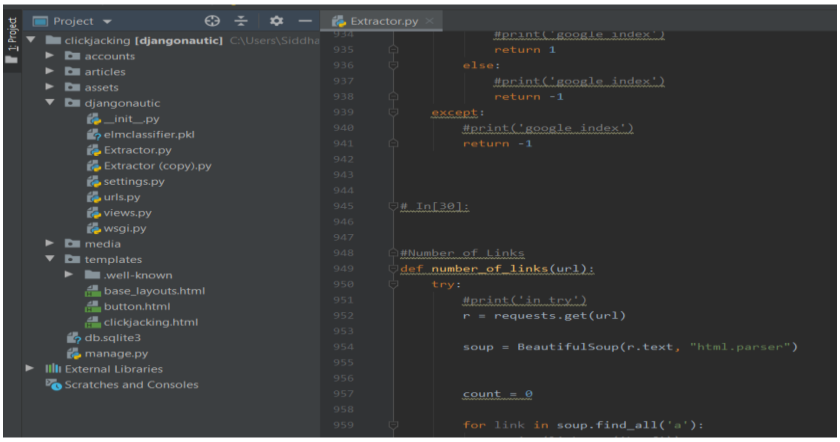

Figure 16: Extractor.py file

Step 10: The following Fig shows the code for the URL extraction from the webpage using the beautifulsoup scrapping technique.

Figure 17: URL Extraction

Step 11: The fig 18 and 19 shows the feature extraction process and the vector representation of features in array format.

```
def ip_feature(url):
    try:
        newurl = (url.split('//')[1]).split('.')[0]
        if newurl.isdigit():
            return 0
        else:
            return 1
    except:
        return -1


# In[3]:


#URL Length
def url_length(url):
    for url in url:
        if len(url)<54:
            #print('url length')
            return 1
        elif len(url)>54 and len(url)<75:
            #print('url length')
            return 0
        else :
            #print('url length')
            return -1
```

Figure 18 : Feature extraction

```
def completefeature(url): #vector represenation
    url_features = [ip_feature(url), url_length(url), having_at_symbol(url),
                    prefix_suffix(url), having_sub_domain(url),
                    sslfinal_state(url), domain_registeration_length(url), favicon(url),
                    port(url), request_url(url), url_of_anchor(url), links_in_tags(url),
                    sfh(url), submitting_to_email(url), on_mouseover(url),
                     age_of_domain(url), dnsrecord(url),
                    web_traffic(url), page_rank(url), google_index(url), number_of_links(url), statistical_report(url)]
    return url_features
```

Figure 19 : Vetor Representation

Step 12: The will classifies the links as phishing and non-phishing

```
def predictor(model, feature_list, urlist): #classification
    string_out = []
    final_op = []
    for index,lis in enumerate(feature_list):
        website = np.array(lis)
        website = website.reshape(1,-1)
        if model.predict(website)[0] == 1:
            string_out.append('Not Phishing')
        else:
            string_out.append('Phishing')
            final_op.append(index)
```

Figure 20 : ELM Classification

Step 13: The HTML file contain in the soup are converted to the string in order to perform further operation on it.

```
#Changing soup
soup = str(soup)  #converted into string format

# adding load static command in the beginning
command ='''{% load static %}'''
soup = command + soup
```

Figure 21 : Soup converted to String format

Step 14: The Iframe links are updated by assigning new values such as margin value: left 150px which are classified as phishing iframe links.

```
def ShiftIframe(soup,url_phishing_list): #IH

    # for high lighting the link
    changetext = 'style="background-color: red;'

    # fetching all the irframe content from a page
    #iframe_sent_lis = re.findall("<iframe[^>]*>(.*?)</iframe[^>]*>",soup)
    iframe_sent_lis = re.findall("<iframe[^>]*>",soup)

    for iframe_Sent in iframe_sent_lis:

        if isHave(url_phishing_list,iframe_Sent):

            # setting condition variable
            flag ="not done"

            print(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>")
            print("Iframe Sentence",iframe_Sent)
            print("<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<")
            # creating copy of iframe_sent
            modified_text = iframe_Sent

            # finding margin-top or margin-left and modified its content
            find_margin = re.findall('margin-top: \d+[a-z]*;|margin-left: \d+[a-z]*;', iframe_Sent)


            for margin_val in find_margin:

                if flag=="not done":
                    modified_text = modified_text.replace(margin_val, "left: 150px;")
```

Figure 22: CSS property are assigned for the phishing Iframe Links.

Step 14: Opacity is also updated.

```python
# changing opacity to maximum
def ChangeOpacity(soup): #oh
    # finding all the text where opacity is mention
    match_op = re.findall('opacity: \d+', soup)
    for text in match_op:
        # replacing all the opacity with 100 %
        print(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>")
        print("Old Opacity:",text)
        print("<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<")
        print(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>")
        print("New Opacity:opacity: 100")
        print("<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<")

        soup = soup.replace(text, 'opacity: 100')
    return soup
```

Figure 23: Opacity is updated

Step 15: Finally, the new updated changes are written in new HTML file, to show the changes.

```python
    with open("./templates/clickjacking.html", "w", encoding='utf', errors='ignore') as file:
        file.write(soup)

        return "DONE"
```

Figure 24: New HTML page is created.